UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

AD-A215 318

DTIC ELECTE NOV 21 1989 B D

1b. RESTRICTIVE MARKINGS

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release;
ditribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
AFOSR-TR- 89-1305

6a. NAME OF PERFORMING ORGANIZATION
Oregon Graduate Center
Department of CS & E

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
AFOSR/NM

6c. ADDRESS (City, State, and ZIP Code)
Baeverton, OR 97006

7b. ADDRESS (City, State, and ZIP Code)
Building 410
Bolling, AFB DC 20332-6448

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
AFOSR

8b. OFFICE SYMBOL (If applicable)
NM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
AFOSR 87-0064

8c. ADDRESS (City, State, and ZIP Code)
Building 410
Bolling, AFB DC 20332-6448

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
61102F	2304	A7	

11. TITLE (Include Security Classification)
Constructive Negotiation in Logic Programs

12. PERSONAL AUTHOR(S)
Richard Hamlet

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
FINAL	FROM Feb87 TO Feb88	March 1988	3

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Logic programming is declarative, but its programs can be executed relatively efficiently. This balance is a precarious one: languages with a more imperative nature are much faster in execution, but programming is more difficult; if the declarative expressiveness of the language is extended, its execution can become so slow that it is unusable. The languages typified by "pure" PROLOG strike this balance on the side of efficiency, by fixing on SLD resolution as the execution algorithm. The Horn-clause subset of first-order logic for which SLD resolution is adequate is limited in the naturalness of its expressiveness, and the most notable omission is that negative information cannot be expressed.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Dr. Abraham Waksman	(202) 767- 5027	NM

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

89 11 17 047

UNCLASSIFIED

Final Report

AFOSR Grant 87-0064
Constructive Negation in Logic Programs

Richard Hamlet
Department of Computer Science and Engineering
Oregon Graduate Center
Beaverton, OR 97006
(503) 690-1153

Summary

This grant supported the final year of Clifford Walinsky's dissertation research. Dr. Walinsky's thesis, with the same title as this grant, has been accepted by OGC, and he is now employed as Assistant Professor of Computer Science by Dartmouth College, Hanover, NH. His thesis work is of high quality, and the ideas it embodies will be important in the emerging area of logic-programming negation.

1. The Problem of Negation in Logic Programming

Logic programming is declarative, but its programs can be executed relatively efficiently. This balance is a precarious one: languages with a more imperative nature are much faster in execution, but programming is more difficult; if the declarative expressiveness of the language is extended, its execution can become so slow that it is unusable. The languages typified by "pure" PROLOG strike this balance on the side of efficiency, by fixing on SLD resolution as the execution algorithm. The Horn-clause subset of first-order logic for which SLD resolution is adequate is limited in the naturalness of its expressiveness, and the most notable omission is that negative information cannot be expressed.

The most common extension of PROLOG incorporating negation is the idea of *negation by failure* (NbF). NbF is a "non-logical" extension to Horn clauses, because its operational meaning is logically incorrect, and programs using it therefore can surprise their authors with unexpected behavior. But it has a further, more serious drawback: it compromises the basic paradigm of logic-programming computing. In a pure PROLOG program, computation proceeds by instantiation of variables in predicates, so that a successful predicate always yields the values that demonstrate its success. When NbF is added, this constructive character is lost.

At the other extreme from NbF, adding a true logical negation to PROLOG would require execution by a general-purpose theorem prover, whose speed can be arbitrarily slower than SLD resolution. A number of compromise proposals have been introduced, but the best of them (in MU Prolog, for example) attack only the logical incorrectness of NbF, not its failure to provide constructive demonstrations.

2. Constructive Negation

Clifford Walinsky noted a device long used by programmers to get around the lack of negation in PROLOG. Faced with the necessity for defining (say) `RelatedByMarriage(X,Y)`, and needing its negation ¬`RelatedByMarriage(X,Y)`, the programmer can sometimes invent a "negative predicate", say `UnrelatedByMarriage`, and define it (positively) in a way that is just the dual of its positive counterpart. When this is possible, the result is a regular PROLOG program that can be efficiently executed by SLD resolution (in fact, by the usual interpreter without change), and that fully retains the constructive nature of execution. Walinsky explored this idea: the circumstances under which the trick works, and its automation so that the programmer need write only the positive form of the predicate (like `RelatedByMarriage`) and have the "negative" form (`UnrelatedByMarriage`) implicitly defined as the meaning of ¬`RelatedByMarriage`.

The best that can be expected from a foray into logic-programming negation is partial success, since if true negation were implemented, the resulting power would require substantially slower execution techniques. By holding to minimal modifications in the execution procedure, Walinsky discovered a clear (if limited) way through the welter of possible extensions to PROLOG. He defined constructive negation as the generalization of the programming trick described above, found conditions under which it is logically sound, and implemented automatic transformation rules and the checks for its validity.

Transformation of positive predicates into their negative duals can introduce universal quantification in the body of the defining clause, so Walinsky had to attack this difficult problem as well. Constructive implementation of universal quantifiers in general requires unbounded searches, but in an important subcase implementation is practical. Frequently, the universally quantified formula is an implication, allowing the antecedent to "filter" instantiations of the consequent. He modified the SLD resolution procedure to handle this situation, and explored circumstances under which the resulting executions are practical.

3. Specific Accomplishments

Cliff Walinsky's thesis lays solid groundwork for systematic study of negation in logic programming that preserves the declarative nature of the languages like pure PROLOG, can be efficiently executed without major changes to present interpreters, and allows programs to retain their constructive solutions. The work is a model of defining a significant research topic, exploring and changing it as more information is gathered. and successfully obtaining results of the kind sought.

AFOSR support not only made possible the research described above, but also provided for extra opportunities for Walinsky. He was invited to attend a Washington, DC workshop as student observer. He was able to attend two major conferences (one an international gathering in Australia which otherwise would have been beyond possibility), to meet and interact with other researchers (and to thereby interest others in his ideas), and ultimately to obtain employment in a prestigious institution where he will continue to be a productive researcher. Several important logic-programming theoreticians visited OGC and provided contact with the broader research community. J-L. Lassez and J. Jaffar are at the core of the logic-programming group at IBM (Yorktown); Lassez invited Cliff to join that group as a visitor in the near future. R. Topor (Melbourne) provided some invaluable early direction for Cliff's work.

The one area in which Cliff was not as successful as we had hoped was in publishing his ideas. He submitted papers to both logic-programming conferences, but they were not accepted. The field is new, not always well defined, and very competitive. There is a measure of politics in writing acceptable papers, particularly for conferences with a high rejection rate. (We were able to learn that the difficulty is not in the ideas, which should find acceptance in the more controlled arena of professional journals.)

The technical reports being submitted under separate cover give a complete description of constructive negation:

Walinsky, C., "Constructive Semantics for Negation", Report CS/E 87-008, Oregon Graduate Center, September, 1987.

Walinsky, C., "Constructive Negation in Logic Progr[illegible], Report CS/E 87-009, Oregon Graduate Center, September, 1987 (PhD thesis).